[illegible]

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/902,818	07/10/2001	Paul F. Ringseth	MSFT-0670/180589.1	9208

7590 09/27/2004

Thomas E. Watson
Woodcock Washburn Kurtz
Mackiewicz & Norris LLP
One Liberty Place- 46th Floor
Philadelphia, PA 19103

EXAMINER	
CHOW, CHIH CHING	

ART UNIT	PAPER NUMBER
2122	

DATE MAILED: 09/27/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

PTO-90C (Rev. 10/03)

[illegible]

Office Action Summary	Application No.	Applicant(s)	
	09/902,818	RINGSETH ET AL.	[illegible]
	Examiner	Art Unit	
	Chih-Ching Chow	2122	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, ma, reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *10 July 2001*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-42* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-42* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application (PTO-152)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

Claim Objections

1. Claims 2, 27 and 29 are objected to because of the following informalities: these claims didn't end with a period; appropriate correction is required.

2. In the specification, paragraph 9, "as networked computing environments continue to push the limits, it would be desirable to combine flexible declarative capabilities with a programming language when implementing SOAP-based communications.", paragraph 10, "In one embodiment, attributes for Visual C++ are utilized as a framework for a declarative syntax for SOAP-based Web services, which Visual C++ attributes have access to type and marshaling information via an attribute provider.", paragraph 33, "the present invention provides means to convert programming language constructs, e.g., C++ programming language constructs, into SOAP messages via a compile-time interaction." – these sentences imply that no specific language should be limited for the current invention, a C++ implementation is only a design choice. However, the following claims contain C++ syntax:
Claim 1 "via a construct of said programming language",
Claim 2 "C++ Construct",
Claim 14 "soap-handler attribute, a soap-method attribute and a soap-header attribute",
Claim 15 "a soap-handler attribute, a soap-method attribute and a soap-header attribute are declared in the at least one Web service's class declaration",
Claim 19 "attribute block",
Claim 36 "a corresponding programming language construct",
Claim 38 "a corresponding programming language construct", and
Claim 39 "at least one programming language construct".

Even C++ examples are given throughout the current invention, the examiner still assumes all the context referred in the above claims that are C++ related should be ignored, and the concepts behind the language are the real disclosures of the current invention.

Claim Rejections - 35 USC § 112

3. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

4. Claims 1, 8, 11 are rejected since the 'communicating with a provider object' is not clearly defined. The 'provider object' is not described in the specification; however, it repeatedly mentioned 'attribute provider'. Examiner herein assumes that the 'provider object' is an object that provides attribute values and it has access to attribute type and marshaling information.

5. *Claim Rejections - 35 USC § 102*

(e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. Claims 1-6, 8, 11, 16-18, 36-42 are rejected under 35 U.S.C. 102(e) as being anticipated by Gunnar Mein, U.S. Pat. No. 6,782,542 (hereinafter "Mein").

CLAIM	**Mein**
1. A method for implementing (Simple Object Access Protocol) SOAP-based Web services via a programming language in a computing system, comprising:	Mein has disclosed all the aspects of using a SOAP-based Web services via a programming language. For item 1, Mein's invention titled "**Simple Object Access Protocol**" (**SOAP**), on column 3, 2nd paragraph, "The inventive protocol includes a data structure which encodes, as a **SOAP** request, **the name of the Automation object of interest, a method to invoke in that object**, and **any valid Automation [in][out] parameters to be exchanged with the object, and creates a client-side SOAP proxy for the Automation object**."
(i) in connection with code that implements at least one SOAP-based Web service, declaring at least one SOAP handling mechanism corresponding to at least one SOAP-based Web service via a construct of said programming language;	For (i), see Mein, Claim 11, "A method of transferring a <u>computer program product</u> from a first computer to a second computer connected to the first computer through a communications medium". – There is no programming language limitation specified in Mein's invention. In order for Mein's invention to work, it has to have a mechanism to <u>declare</u> that SOAP is the designated communication protocol. Mein

discloses 'SOAP handling mechanism', see Mein, column 5, lines 24-62, "SOAP is a data transmission paradigm. The data transmission paradigm includes a three-section data structure that comprises a **header**, body, and trailer. The data structure is used to package information referring to a **request to invoke a method of an Automation object**. In operation when the client process requires certain data from an Automation object (***provider object***, since the Automation object is providing data to the client) the process issues a method call, which causes an Advanced DataSpace to be created. The Advanced DataSpace, in turn, creates a SOAP proxy packages the data structure as an HTTP POST message in multipart MIME packets, and sends the message as a binary data stream through the network, i.e., the Internet to the server computer where the Automation object is located. When the server computer receives the HTTP POST message, the server process, i.e., the Web server, invokes a SOAP stub (***SOAP handling mechanism***) for the SOAP proxy. The SOAP stub that is invoked is chosen based on an identifier contained in the header of the data structure. The SOAP stub unpackages the multipart MIME packets and instantiates the Automation object identified in the header of the data structure. A method name field also identified in the header of the data structure indicates the method of the Automation object to invoke. **The method is invoked by the SOAP stub using [in] parameters contained in the body of the data structure. After the method has finished executing, return, or [out], parameters are returned to the SOAP stub, which packages the [out] parameters as multipart MIME packets and transmits a resulting HTTP Response message as a**

(ii) when compiling said code, communicating with a provider object; and (iii) generating at least one of additional code and data for use at run-time when at least one of sending and receiving a SOAP message for said at least one SOAP-based Web service occurs.	**binary data stream across the Internet to the SOAP proxy** (item (iii), **sending receiving SOAP-based web services**). The SOAP proxy unpackages the multipart MIME packets and returns the [out] and [in, out] parameters to the client process. The instance of the Automation object is reclaimed after the [out] parameters are returned to the SOAP stub. The [out] parameters, like the [in] parameters, are contained in the body of the data structure"; In Mein's method, 'SOAP' is designated as the communication protocol, or else it's not going to be treated as a SOAP-based Web service. (ii)The computer program product has to be compiled in order to make it executable, and additional code and data is hence produced after the compilation. (iii) Once the declaration of a SOAP message is done, SOAP-based message can be created, sent, received and processed in both SOAP-based Web sides. In Mein, column 3, 4th paragraph, "the SOAP stub, which is **running** on the Web server, unpacks and parses the SOAP request, instantiates the COM Automation object, and invokes the method with the marshaled [in] parameters." **(run-time feature).**
2. A method according to claim 1, wherein said declaring includes declaring at least one SOAP handling mechanism corresponding to at least one SOAP-based Web service via a C++ construct;	Same as claim 1 rejection (since Mein does not have any language limitation, therefore Mein covers wider range than current invention).
3. A method according to claim 2, wherein said declaring includes declaring at least one C++ attribute corresponding to at least one SOAP-based Web service.	Same as claim 2 rejection.

4. A method according to claim 1, wherein when sending a SOAP message for said at least one SOAP-based Web service, the method further includes utilizing said at least one of additional code and data at run-time to generate the SOAP message.	For the features of claim 1 see claim 1 rejection. For the rest of the claim 4 features, see Mein, column 3, 4th paragraph, "the SOAP stub, which is **running** on the Web server, unpacks and parses the SOAP request, instantiates the COM Automation object, and invokes the method with the marshaled [in] parameters." **(run-time feature)**
5. A method according to claim 1, wherein when receiving a SOAP message for said at least one SOAP-based Web service, the method further includes utilizing said at least one of additional code and data at rum-time to convert said SOAP message to an object of the programming language.	Same as claim 1 rejection.
6. A method according to claim 1, wherein said declaring reduces the amount of code a developer writes relative to writing the code without said declaring.	Same as claim 1 rejection. Mei must have a way to declare a SOAP-based protocol, because without 'declaring' the SOAP as the designated protocol, it can't use a SOAP-based communication protocol. The 'said declaring' in claim 1 is not clearly specified HOW it's different from other people's declaration or how it reduces code.
8. A method according to claim 1, wherein said provider object communicates with the compiler of said code to generate said at least one of additional code and data.	Same as claim 1 rejection.
11. A method according to claim 1, wherein said provider object is an attribute provider object and said attribute provider object has access to attribute type and marshaling information.	For the features of claim 1 see claim 1 rejection (see 112 (2) rejection 3). For the rest of the claim 4 features, see Mein, column 3, 4th paragraph, "The SOAP proxy **marshals** and transfers the multipart MIME-encoded SOAP request to an Applications Programming Interface (API) which acts as a server-side SOAP stub for processing SOAP messages. **Marshaling** is the

	process of packaging up the data so that when it is sent from one process to another, the receiving process can decipher the data."
16. A computer readable medium bearing computer executable instructions for carrying out the method of claim 1.	Same as claim 1 rejection.
17. A modulated data signal carrying computer executable instructions for performing the method of claim 1.	Same as claim 1 rejection.
18. A computing device comprising means for performing the method of claim 1.	Same as claim 1 rejection.
36. A method processing (Simple Object Access Protocol) SOAP messages at run-time in a computing system having a server with at least one application having at least one user object implementing at least one SOAP-based Web service, comprising: at least one of receiving an incoming SOAP message from a client computing device and receiving a call from a user object with at least one programming language construct; and with a runtime intermediate layer, at least one of converting said incoming SOAP message to a corresponding programming language construct and converting said call with at least one programming language construct to an outgoing SOAP message, wherein said runtime intermediate layer utilizes at least one of code and data generated during compilation of said user object.	Same as claim 1 rejection.
37. A method according to claim 36, wherein said programming language is C++.	For the features of claim 36 see claim 36 rejection. For the rest of the features of claim 37 see claim 2 rejection.

38. A method according to claim 36, wherein said converting of said incoming SOAP message to a corresponding programming language construct includes determining the appropriate user object to receive the SOAP message.	For the features of claim 36 see claim 36 rejection. Again, the converting and determining the appropriate user object is in Mein, column 5, lines 47-62, "A method name field also identified in the header of the data structure indicates the method of the Automation object to invoke. The method is invoked by the SOAP stub using **[in] parameters** contained in the body of the data structure. After the method has finished executing, return, or **[out], parameters are returned** to the SOAP stub, which packages the [out] parameters as multipart MIME packets and transmits a resulting HTTP Response message as a binary data stream across the Internet to the SOAP proxy. The SOAP proxy unpackages the multipart MIME packets and returns the [out] and [in, out] parameters to the client process. The instance of the Automation object is reclaimed after the [out] parameters are returned to the SOAP stub. The [out] parameters, like the [in] parameters, are contained in the body of the data structure" ***converting input data into output data via predefined process, and determining the user object***.
39. A method according to claim 36, wherein said converting said call with at least one programming language construct to an outgoing SOAP message includes formatting the message for delivery to an intended client computing device.	For the features of claim 36 see claim 36 rejection. Again, in Mein, column 3, lines 40-42, "any valid Automation **[in][out] parameters to be exchanged** with the object, and creates a client-side SOAP proxy for the Automation object"**;** in Mein, column 47-62, "The method is invoked by the SOAP stub using **[in] parameters** contained in the body of the data structure. After the method has finished executing, return, or **[out], parameters** are returned to the SOAP stub, which **packages the [out] parameters as multipart MIME packets** and transmits a resulting HTTP Response

	message as a binary data stream across the Internet to the SOAP proxy." this process includes **converting** and **formatting** the message for delivery to an intended client computing device.
40. A computer readable medium bearing computer executable instructions for carrying out the method of claim 36.	For the features of claim 36 see claim 36 rejection. For the rest of the features of claim 40 see claim 1 (iii) rejection.
41. A modulated data signal carrying computer executable instructions for performing the method of claim 36.	For the features of claim 36 see claim 36 rejection. For the rest of the features of claim 41 see claim 4 rejection (run-time feature).
42. A computing device comprising means for performing the method of claim 36.	For the features of claim 36 see claim 36 rejection. For the rest of the features of claim 42 see claim 1 rejection.

Claim Rejections - 35 USC § 103

7. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> **(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.**

8. Claims 7, 9, 10, 12-15 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gunnar Mein, U.S. Pat. No. 6,782,542 (hereinafter "Mein"), further in view of "Mapping CORBA and SOAP", 10/31/2000 (hereinafter "CORBA").

CLAIM	Mein / CORBA
7. A method according to claim 1, wherein said declaring includes describing at least one Web service interface using embedded interface definition language (IDL).	For the features of claim 1 see claim 1 rejection. Mein has taught all the aspects in claim 7, except the 'IDL' part. However, CORBA shows the feature in an analogous art. On CORBA, page 7, under XORBA example has shown using IDL on a SOAP-based Web service interface. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement of the SOAP-based Web service with the Interface Definition Language (IDL) further taught by CORBA for the purpose of transmitting objects via a SOAP-based protocol via the internet (see CORBA page 6, 2nd paragraph).
9. A method according to claim 1, wherein via said declaring, said method hides from the developer the underlying details regarding the SOAP protocol, dispatching to the appropriate object and function, marshaling the (extensible Markup Language) XML, un-marshaling the XML, and generating the SOAP response.	For the feature of claim 1 see claim 1 rejection. Mein also discloses the feature of 'marshaling and un-marshaling the data, see Mein's Abstract, "The format essentially encodes the automation object's name, method to invoke, and any [in], [out], [in, out] parameters that the method signature requires, packages them up into a custom MIME type and marshals it to the ISAPI dynamic link library (DLL) on the IIS/HTTP server. There, the ISAPI DLL contains the logic to unpack the SOAP request, parses it, creates the Automation object, invokes the method with the marshaled parameters, and then returns any [out] parameters to the caller/client using the SOAP protocol." Mein didn't specifically mentioned XML in his art. However, CORBA shows the XML in an analogous art, see CORBA page 1, last sentence, "SOAP (Simple Object Access Protocol) transmits business data expressed in the eXtensible Markup Language (XML) over the widely-used web protocol HTTP." It would have been obvious to a person of ordinary skill in the art at the time of the

	invention was made to supplement of the marshaling/un-marshaling functions with the extensible Markup (XML) further taught by CORBA for the purpose of transmitting objects via a SOAP-based protocol via the internet (see CORBA page 6, 2nd paragraph).
10. A method according to claim 1, wherein the underling XML packaging of the SOAP message is transported according to said at least one of sending and receiving via at least one of hypertext transfer protocol (HTTP), file transfer protocol (FTP), transmission control protocol (TCP), user datagram protocol (UDP), internet relay chat (IRC), telnet protocol and Gopher protocol.	For the features of claim 1 see claim 1 rejection. The claim 9 rejection has covered the XML feature. For the rest of the features of claim 10 see claim 7 rejection, the claim said 'at least' one of the listed protocols, the XORBA example was given for claim 7 rejection uses **HTTP**.
12. A method according to claim 1, wherein for each method of said code that is to be exposed as part of the at least one Web service, the method is introduced via an interface as part of an interface definition language (IDL) description.	For the features of claim 1 see claim 1 rejection. For the rest of the features of claim 12 see claim 7 rejection.
13. A method according to claim 12, wherein said declaring further includes specifying at least one of in, out, size-is and retval IDL attributes in the at least one Web service's interface declaration.	For the features of claim 12 see claim 12 rejection. For the rest of the features of claim 13 see claim 7 rejection. The XORBA example on CORBA page 7 has 'in' parameters.
14. A method according to claim 1, wherein said declaring includes declaring at least one of a soap-handler attribute, a soap-method attribute and a soap-header attribute.	For the features of claim 1 see claim 1 rejection. For the rest of the features of claim 14 see claim 7 rejection. The XORBA example on CORBA page 7 has a 'SOAP:Header'.
15. A method according to claim 14, wherein said at least one of a soap-handler attribute, a soap-method	For the features of claim 14 see claim 14 rejection. For the rest of the features of claim 15 see claim 7 rejection. (The XORBA

attribute and a soap-header attribute are declared in the at least one Web service's class declaration.	example which has a soap-header declared in HTTP.)

9. Claims 19, 21, 22, 33-35 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gunnar Mein, U.S. Pat. No. 6,782,542 (hereinafter "Mein"), further in view of "Introducing SOAP" by Reuven M. Lerner, "Linux Journal", March 2001 (hereinafter "Lerner").

CLAIM	**Mein / Lerner**
19. A method of compiling programming language code with a compiler, comprising: (i) initially parsing the code; (ii) during said initial parsing, encountering an attribute block and allowing an interface definition that follows the attribute block to include embedded information ; and (iii) parsing the interface definition.	For (i) and (iii), Mein's disclosure about header, body, and trailer data structures all need to be '**parsed** and **compiled**', the compiler process (including parsing) is inherited from any computer program operation. Mein has disclosed all the features in claim 19 except the 'encountering an attribute block and allowing an interface definition that follows the attribute block to include embedded information' part specifically (actually it can all be embedded in his 'SOAP stub' process). However, (ii) Lerner shows the 'encountering an attribute block and allowing an interface definition that follows the attribute block to include embedded information' feature in an analogous art, in

	information' feature in an analogous art, in Lerner, page 6, 7th paragraph, "The XML itself begins with an XML declaration and then a SOAP envelope. Inside the envelope is an optional header and a mandatory body. The body names the object and method that we wish to invoke, as well as any arguments that we might have passed. This XML is **parsed** into the native operating system and language format and is then passed along to the target object. The object returns a response value to the SOAP server which then creates a SOAP response in XML as seen in Listing 4." Here the 'object and method' are the **embedded information** and the 'arguments' contains the **interface definition**. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement of the SOAP declarations with the SOAP envelope further taught by Lerner for the purpose of transmitting SOAP request via the internet (see Lerner page 2, 5th paragraph).
21. A method according to claim 19, further including determining whether said attribute block includes an object attribute.	For the features of claim 19 see claim 19 rejection. For the rest of the features of claim 21 see claim 1 rejection (SOAP handling mechanism).
22. A method according to claim 21, wherein for each function in the interface definition, the compiler parses the function declaration, and saves the parameter attribute information for use later in determining a (Simple Object Access Protocol) SOAP message structure.	For the features of claim 21 see claim 21 rejection. For the rest of the features of claim 22 see claim 1 rejection (SOAP handling mechanism).
33. A computer readable medium bearing computer executable instructions for carrying out the method of claim 19.	Same as claim 19 rejection.

34. A modulated data signal carrying computer executable instructions for performing the method of claim 19.	Same as claim 19 rejection.
35. A computing device comprising means for performing the method of claim 19.	Same as claim 19 rejection.

10. Claim 20 is rejected under 35 U.S.C. 103(a) as being unpatentable over Gunnar Mein, U.S. Pat. No. 6,782,542 (hereinafter "Mein"), further in view of "Mapping CORBA and SOAP", 10/31/2000 (hereinafter "CORBA"), and further in view of "Introducing SOAP" by Reuven M. Lerner, Linux Journal, March 2001 (hereinafter "Lerner").

CLAIM	**Mein / CORBA / Lerner**
20. A method according to claim 19, wherein said embedded information is implemented via interface definition language (IDL).	For the features of claim 19 see claim 19 rejection, which includes Mein's SOAP handling mechanism and Lerner's embedded information processing. For the IDL feature of claim 20 see claim 7 rejection. It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement of the SOAP mechanism (Mein), SOAP envelope (embedded information, Lerner), by IDL further taught by CORBA for the purpose of transmitting SOAP request via the internet (see CORBA page 6, 2nd paragraph).

11. Claims 23-32 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gunnar Mein, U.S. Pat. No. 6,782,542 (hereinafter "Mein"), further in view of "Introducing SOAP" by Reuven M. Lerner, Linux Journal, March 2001 (hereinafter "Lerner"), and further in view of Aaron Skonnard "SOAP: The Simple Object Access Protocol", Januray, 2000 (hereinafter "Skonnard").

CLAIM	**Mein / Lerner / Skonnard**
23. A method according to claim 19, wherein when the compiler encounters a soap-handler attribute, the compiler recognizes soap-handler as an attribute that is supported by an attribute provider and calls into the attribute provider.	For the features of claim 19 see claim 19 rejection. For the rest of the features of claim 23 see claim 1 rejection (SOAP handling mechanism). Mein and Lerner have taught all the aspects in claim 23, except the 'calls into the attribute provider' part. However, Skonnard shows the feature in an analogous art. On Skonnard, page 6 under 'SOAP XML Payload', "For a SOAP request, the XML payload consists of several elements including a root element, a method element (referred to as the call element)," page 7, "The **element must contain an id attribute** that distinguishes it from other child elements of the root SerializedStream element. The **call element** also contains a child element for each [in] and [in/out] parameter for the given method." The **call element** performs same function as **calls into the attribute provider**. Further in page 13, "This CGI script simply uses SOAP::Dispatcher to process the SOAP request. The $classes variable passed to handle_cgi_post lets you to specify the class names that are allowed to be **called from the outside world**— this adds an additional layer of application security. Remember, using SOAP you can call this Perl object from any client on any platform using any language that supports HTTP and XML. The Perl language binding

	makes it especially easy to make SOAP <u>method</u> calls from Perl clients." It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement of the SOAP-based Web service (Mein) and SOAP envelope (Lerner) with the calls into the attribute provider further taught by Skonnard for the purpose of making SOAP soap-handler calls (see Skonnard page 13 last paragraph).
24. A method according to claim 23, wherein the attribute provider marks the class on which the soap-handler attribute appears as handling (Simple Object Access Protocol) SOAP messages and then the attribute provider returns control to the compiler.	For the features of claim 23 see claim 23 rejection. For the rest of the features of claim 24 see claim 1 rejection (SOAP handling mechanism).
25. A method according to clam 24, wherein after said return of control, the compiler begins parsing the body of the class on which the soap-handler attribute appears and parses each function in the class.	For the features of claim 24 see claim 24 rejection. For the rest of the features of claim 25 see claim 1 rejection (SOAP handling mechanism).
26. A method according to 25, wherein for each soap-header attribute on the function, the compiler performs soap-header attribute processing.	For the features of claim 25 see claim 25 rejection. Mein discloses the 'soap-header' feature (see claim 1 rejection), Skonnard further shows the feature of 'soap-header', see page 4, under 'SOAP HTTP Headers', "SOAP defines this value to identify an HTTP entity-body containing an XML-encoded <u>method</u> invocation or response. All SOAP requests and responses must use this Content-Type value to be recognized. SOAP defines three extended HTTP headers. Two of these headers, MessageType and MethodName, must be used with every SOAP request. The third header, InterfaceName, is optional. The MessageType header defines whether the

	given HTTP body contains a method call or a method response and must contain the values Call or CallResponse, respectively. The MethodName header defines the name of the method to invoke. The optional InterfaceName header defines the name of the interface to which MethodName belongs. Because of HTTP's extensibility characteristics, you can create custom headers for application-specific purposes without any problem." It would have been obvious to a person of ordinary skill in the art at the time of the invention was made to supplement of the SOAP-based Web service (Mein) and SOAP envelope (Lerner) with soap-header process further taught by Skonnard for the purpose of making SOAP soap-handler calls (see Skonnard page 13 last paragraph).
27. A method according to claim 26, wherein said soap-header attribute processing includes recognizing a soap-header attribute as an attribute that is supported by the attribute provider and calling into the attribute provider whereby the attribute provider (i) gathers information to create the "Header" part of a SOAP message, (ii) queries the compiler for information about the parameters to the soap-header attribute and (iii) queries the compiler for information about the data type for the header	For the features of claim 26 see claim 26 rejection. For the rest of the features of claim 27 see claim 23.
28. A method according to claim 26, wherein said soap-header attribute processing includes generating the information for creating a portion of the (Simple Object Access Protocol) SOAP message header and returning control from the attribute provider to the compiler.	For the features of claim 26 see claim 26 rejection. For the rest of the features of claim 28 see claim 23 rejection.

29. A method according to 25, wherein if the function has a soap-method attribute, the compiler performs soap-method attribute processing	For the features of claim 25 see claim 25 rejection. For the rest of the features of claim 29 see claim 23 rejection.
30. A method according to claim 29, wherein said soap-method attribute processing includes recognizing a soap-method attribute as an attribute that is supported by the attribute provider and calls into the attribute provider whereby the attribute provider (i) gathers information to create a (Simple Object Access Protocol) SOAP message for the function and (ii) queries the compiler for information about the parameters to the function on which soap-method attribute appears.	For the features of claim 29 see claim 29 rejection. For the rest of the features of claim 30 see claim 23.
31. A method according to claim 30, wherein for each parameter on the function, the attribute provider (i) uses the interface definition information to determine which function parameters are to be received as part of the incoming SOAP message, and which parameters are to be sent back as part of the SOAP message response and (ii) queries the compiler about the data type of the parameter.	For the features of claim 30 see claim 30 rejection. For the rest of the features of claim 31 see claim 23 rejection (in out parameters).
32. A method according to claim 29, wherein said soap-method attribute processing includes generating the data to create the SOAP message portion for a parameter and returning control from the attribute provider to the compiler.	For the features of claim 29 see claim 29 rejection. For the rest of the features of claim 32 see claim 1 rejection (SOAP handling mechanism).

Conclusion

12. The following summarizes the status of the claims:

Claim objections: 1, 2, 14, 15, 19, 27, 29, 36, 38, and 39
35 USC § 112 rejections: 1, 8, and 11
35 USC § 102 (e) rejections: 1-6, 8, 11, 16-18, 36-42
35 USC § 103 rejections: 7, 9-10, 12-15, 19-35

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Chih-Ching Chow whose telephone number is 703-305-7205. The examiner can normally be reached on 7:00am - 3:30pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Dam can be reached on 703-305-4552. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Chih-Ching Chow
Examiner
Art Unit 2122

CC

[illegible]

JOHN CHAVIS
PATENT EXAMINER
ART UNIT 2124